APPENDIX 4

```
/**
* IPS VPN Tunnel update functions
* @file update_tunnels.h
* @author jmccaskey
*/

#ifndef UPDATE_TUNNELS__H
#define UPDATE_TUNNELS__H

#include "snmpwalk.h"

/**
* Function to update all tunnel related ids for a specific device.
* The function will find any tunnels involving the device using the passed MySQL connection  and
* then perform the neccesary walking and matching to update the tunnel index ids
* in the database.
*/
int update_tunnels(struct ips_device *device, MYSQL *mysql_connection);

#include "update_tunnels.c"

#endif
```

```
/**
* IPS VPN Tunnel update functions
* @file update_tunnels.c
* @author jmccaskey
*/

/**
* Function to update all tunnel related ids for a specific device.
* The function will find any tunnels involving the device using the passed MySQL connection  and
* then perform the neccesary walking and matching to update the tunnel index ids
* in the database.
*/
int update_tunnels(struct ips_device *device, MYSQL *mysql_connection) {
        MYSQL_RES *result;
        MYSQL_ROW row;
        char *sql_query;
        int n;

        assert(sql_query=malloc(2500));
        n=snprintf(sql_query, 2500, "SELECT tunnel.tunnel_id, tunnel.tunnel_server_id,
tunnel_side_one.tunnel_entry_id as tunnel_entry_id_local, "
                        "tunnel_side_one.tunnel_entry_server_id as tunnel_entry_server_id_local, "
                        "tunnel_side_one.tunnel_side_id as tunnel_side_id_local, "
                        "tunnel_side_one.tunnel_side_server_id as tunnel_side_server_id_local, "
                        "tunnel_side_one.sa_id as sa_id_local, tunnel_side_two.tunnel_side_id as
tunnel_side_id_remote, "
                        "tunnel_side_two.tunnel_side_server_id as tunnel_side_server_id_remote,
tunnel_side_two.sa_id as sa_id_remote, "
                        "tunnel_side_two.tunnel_entry_id as tunnel_entry_id_remote, "
                        "tunnel_side_two.tunnel_entry_server_id as tunnel_entry_server_id_remote,
device_one.ip_address as ip_address_local, "
                        "device_one.device_id as device_id_local, device_one.device_server_id as
device_server_id_local, "
                        "tunnel_entry_one.tunnel_entry_type as tunnel_entry_type_local, "
                        "tunnel_entry_two.tunnel_entry_type as tunnel_entry_type_remote, "
                        "device_two.ip_address as ip_address_remote "
                        "FROM tunnel, tunnel_side as tunnel_side_one, tunnel_side as tunnel_side_two,
"
                        "tunnel_entry as tunnel_entry_one, tunnel_entry as tunnel_entry_two, "
                        "device as device_one, device as device_two "
                        "WHERE tunnel_side_one.tunnel_id = tunnel.tunnel_id "
                        "AND tunnel_side_one.tunnel_server_id = tunnel.tunnel_server_id "
                        "AND tunnel_side_two.tunnel_id = tunnel.tunnel_id "
                        "AND tunnel_side_two.tunnel_server_id = tunnel.tunnel_server_id "
                        "AND NOT(tunnel_side_two.tunnel_side_id = tunnel_side_one.tunnel_side_id "
                        "AND tunnel_side_two.tunnel_side_server_id =
tunnel_side_one.tunnel_side_server_id) "
                        "AND tunnel_side_one.tunnel_entry_id=tunnel_entry_one.tunnel_entry_id "
                        "AND
tunnel_side_one.tunnel_entry_server_id=tunnel_entry_one.tunnel_entry_server_id "
                        "AND tunnel_side_two.tunnel_entry_id=tunnel_entry_two.tunnel_entry_id "
                        "AND
tunnel_side_two.tunnel_entry_server_id=tunnel_entry_two.tunnel_entry_server_id "
                        "AND tunnel_entry_one.device_id=device_one.device_id "
                        "AND tunnel_entry_one.device_server_id=device_one.device_server_id "
                        "AND tunnel_entry_two.device_id=device_two.device_id "
```

```
                        "AND tunnel_entry_two.device_server_id=device_two.device_server_id "
                        "AND device_one.device_id = %s "
                        "AND device_one.device_server_id=%s", device->device_id, device-
>device_server_id);
        //execute query
    if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
        flockfile(stderr);
        fprintf(stderr, "%s: Failed executing query for tunnels associated with device: Error: %s\n",
timestamp, mysql_error(mysql_connection));
        funlockfile(stderr);
        free(sql_query);
    }
    free(sql_query);

        //store results from last query into result
        result=mysql_store_result(mysql_connection);
        //setup snmp session options
    session = ips_snmp_sess_init(device);

        index_node *inode = NULL;
        //loop through each tunnel_side updating its ids
        while(row=mysql_fetch_row(result)) {
                int n;
                char *start_oid, *sql_query;
                queue index_queue;
                queue_init(&index_queue);
                /** Check what type of tunnel we are dealing with and call the walk function */
                if(strcmp(row[15], "cisco_ipsec")==0) {
                        //put each digit of the ip address into a seperate array element
                        unsigned char ipaddress[4] = {0, 0, 0, 0};
                        if(row[17] != NULL) {
                                char delim[1] = {'.'};
                                char *pos;
                                pos = row[17];
                                //strsep is a gnu c specific extension... it replaces the non thread safe
(and slower) strtok from ansi c...
                                ipaddress[0] = atoi(strsep(&pos, delim));
                                ipaddress[1] = atoi(strsep(&pos, delim));
                                ipaddress[2] = atoi(strsep(&pos, delim));
                                ipaddress[3] = atoi(strsep(&pos, delim));
                        }
                        assert(start_oid = malloc(200));

                        //walk the phase 1 oid
                        snprintf(start_oid, 200,
"cipSecPhaseOne.cikeTunnelTable.cikeTunnelEntry.cikeTunRemoteAddr");
                        ips_snmpwalk(device, row[15], start_oid, ipaddress, &index_queue, 1);
                        free(start_oid);

                        //process the queue in order to get the phase 1 id
                        int count = 0;
                        while(index_queue.head != NULL) {
                                if(count > 0)
                                        free(inode);
                                inode = (index_node *) queue_get(&index_queue);
                                ++count;
```

```
                        }

                        //update the phase 1 id in db...
                        assert(sql_query = malloc(800));
                        if(count > 0) {
                                n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                                "session_id_one_previous = session_id_one,
session_id_one = %d "
                                                "WHERE tunnel_side_id = %s AND
tunnel_side_server_id = %s",
                                                inode->value, row[4], row[5]);
                        } else {
                                //there was nothing in the queue, so we know we didn't find a matching
session id, insert NULL
                                n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                                "session_id_one_previous = session_id_one,
session_id_one = NULL "
                                                "WHERE tunnel_side_id = %s AND
tunnel_side_server_id = %s",
                                                row[4], row[5]);
                        }
                        free(inode);

                        //flockfile(stdout);
                        //fprintf(stdout, "%s\n", sql_query);
                        //funlockfile(stdout);

                        if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
                                flockfile(stderr);
                                fprintf(stderr, "%s: Failed executing query to update tunnel_side: Error:
%s\n", timestamp, mysql_error(mysql_connection));
                                funlockfile(stderr);
                                free(sql_query);
                        } else {
                                free(sql_query);
                        }

                        assert(start_oid = malloc(200));
                        //walk the phase 2 oid
                        snprintf(start_oid, 200,
"cipSecPhaseTwo.cipSecTunnelTable.cipSecTunnelEntry.cipSecTunRemoteAddr");
                        ips_snmpwalk(device, row[15], start_oid, ipaddress, &index_queue, 0);
                        free(start_oid);

                        //find all cisco phase 2 monitors related to this tunnel side so they can be updated
                        assert(sql_query = malloc(1000));
                        n = snprintf(sql_query, 1000, "SELECT monitor.monitor_id,
monitor.monitor_server_id "
                                "FROM monitor_tunnel, monitor, metric, metric_snmp "
                                "WHERE monitor_tunnel.tunnel_side_id=%s AND
monitor_tunnel.tunnel_side_server_id=%s "
                                "AND monitor.monitor_id=monitor_tunnel.monitor_id "
                                "AND monitor.monitor_server_id=monitor_tunnel.monitor_server_id "
                                "AND metric.metric_id=monitor.metric_id "
                                "AND metric.metric_server_id=monitor.metric_server_id "
                                "AND metric.suite='snmp' "
```

```
                                "AND metric_snmp.metric_id=metric.metric_id "
                                "AND metric_snmp.metric_server_id=metric.metric_server_id "
                                "AND metric_snmp.phase = 'two' ",
                                                row[4], row[5]);
                        if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
                    flockfile(stderr);
                                fprintf(stderr, "%s: Failed executing query to find phase 2 monitors during
tunnel update: Error: %s\n", timestamp, mysql_error(mysql_connection));
                    funlockfile(stderr);
                    free(sql_query);
                } else {
                    free(sql_query);
                }
                        MYSQL_RES *phase_two_result;
                        MYSQL_ROW monitor_row;
                        phase_two_result = mysql_store_result(mysql_connection);

                        while(monitor_row=mysql_fetch_row(phase_two_result)) {
                                char *sql_query_delete;
                                int len;
                                assert(sql_query_delete = malloc(8000));
                                //additional where clauses will be appened as we go to avoid deleting the
rows that are still in use...
                                len = snprintf(sql_query_delete, 8000, "DELETE FROM
monitor_tunnel_cisco_phase_2 WHERE monitor_id=%s AND monitor_server_id=%s", monitor_row[0],
monitor_row[1]);

                                inode = (index_node *) index_queue.head;
                                while(inode != NULL) {
                                        //check if there is already a row for this session id value
                                        assert(sql_query = malloc(1000));
                                        n=snprintf(sql_query, 1000, "SELECT COUNT(*) AS count
FROM monitor_tunnel_cisco_phase_2 "
                                                        "WHERE monitor_id=%s AND
monitor_server_id=%s "
                                                        "AND phase_2_id=%d ", monitor_row[0],
monitor_row[1], inode->value);
                                        if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
                                    flockfile(stderr);
                                  fprintf(stderr, "%s: Failed executing query for phase 2 row count: Error:
%s\n", timestamp, mysql_error(mysql_connection));
                                funlockfile(stderr);
                                    free(sql_query);
                                } else {
                                    free(sql_query);
                                  }
                                        MYSQL_RES *count_result;
                                        MYSQL_ROW count_row;
                                        count_result = mysql_store_result(mysql_connection);
                                        count_row = mysql_fetch_row(count_result);
                                        int row_count = atoi(count_row[0]);
                                        mysql_free_result(count_result);
                                        if(row_count < 1) {
                                                //there is no row for this session id, create one
                                                assert(sql_query = malloc(800));
                                                n=snprintf(sql_query, 800, "INSERT INTO
```

```
monitor_tunnel_cisco_phase_2 "
                                                        "(monitor_id, monitor_server_id,
phase_2_id, counter, timestamp) "
                                                        "VALUES (%s, %s, %d, 0, 0)",
                                                        monitor_row[0], monitor_row[1],
inode->value);
                                                if(mysql_real_query(mysql_connection, sql_query, n)!=0) .
{
                                    flockfile(stderr);
                                        fprintf(stderr, "%s: Failed executing query to insert into
monitor_tunnel_cisco_phase_2: Error: %s\n", timestamp, mysql_error(mysql_connection));
                                      funlockfile(stderr);
                                   free(sql_query);
                                    } else {
                                    free(sql_query);
                                   }
                                            }
                                            //update the delete query to not delete this row (since it still is in
use)
                                            char *temp_string;
                                            temp_string = strdup(sql_query_delete);
                                            len = snprintf(sql_query_delete, 8000, "%s AND
NOT(phase_2_id=%d) ", temp_string, inode->value);
                                            free(temp_string);
                                            inode = (index_node *)inode->next;
                                    }
                                    //execute the delete for all rows for this monitor that were not still in use
                                    if(mysql_real_query(mysql_connection, sql_query_delete, len)!=0) {
                                flockfile(stderr);
                            fprintf(stderr, "%s: Failed executing query to delete old phase 2 rows: Error:
%s\n", timestamp, mysql_error(mysql_connection));
                            funlockfile(stderr);
                            free(sql_query_delete);
                        } else {
                            free(sql_query_delete);
                        }

                            }
                            //cleanup the queue
                            while(index_queue.head != NULL) {
                                    inode = (index_node *) queue_get(&index_queue);
                                    free(inode);
                            }
                            mysql_free_result(phase_two_result);

                     } else if(strcmp(row[15], "netscreen_ipsec")==0) {
                    assert(start_oid = malloc(200));

                    //walk the phase 1 oid
                    snprintf(start_oid, 200, "nsVpnMonTable.nsVpnMonEntry.nsVpnMonSaId");
                    ips_snmpwalk(device, row[15], start_oid, row[6], &index_queue, 1);
                    free(start_oid);

                    //process the queue in order to get the phase 1 id
                    int count = 0;
                            while(index_queue.head != NULL) {
```

```
                        if(count > 0)
                                free(inode);
                inode = (index_node *) queue_get(&index_queue);
                ++count;
        }

        //update the phase 1 id in db...
        assert(sql_query = malloc(800));
        if(count > 0) {
                n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                "session_id_one_previous = session_id_one, session_id_one = %d "
                                "WHERE tunnel_side_id = %s AND tunnel_side_server_id = %s",
                                inode->value, row[4], row[5]);
        } else {
                //there was nothing in the queue, so we know we didn't find a matching session id,
insert NULL
                n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                "session_id_one_previous = session_id_one, session_id_one = NULL
"
                                "WHERE tunnel_side_id = %s AND tunnel_side_server_id = %s",
                                row[4], row[5]);
                }
                free(inode);

                //flockfile(stdout);
        //fprintf(stdout, "%s\n", sql_query);
                //funlockfile(stdout);

        if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
                flockfile(stderr);
                        fprintf(stderr, "%s: Failed executing query to update tunnel_side: Error:
%s\n", timestamp, mysql_error(mysql_connection));
                funlockfile(stderr);
                free(sql_query);
        } else {
                free(sql_query);
        }

          } else if(strcmp(row[15], "altiga_ipsec")==0) {
                assert(start_oid = malloc(200));

        //walk the phase 1 oid
                snprintf(start_oid, 200,
"alActiveSessionTable.alActiveSessionEntry.alActiveSessionIpAddress");
        ips_snmpwalk(device, row[15], start_oid, row[17], &index_queue, 1);
        free(start_oid);

        //process the queue in order to get the phase 1 id
        int count = 0;
        while(index_queue.head != NULL) {
                        if(count>0)
                 free(inode);
                inode = (index_node *) queue_get(&index_queue);
                ++count;
        }
```

```
                //update the phase 1 id in db...
                assert(sql_query = malloc(800));
                if(count > 0) {
                        n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                        "session_id_one_previous = session_id_one, session_id_one = %d "
                                        "WHERE tunnel_side_id = %s AND tunnel_side_server_id = %s",
                                        inode->value, row[4], row[5]);
                } else {
                        //there was nothing in the queue, so we know we didn't find a matching session id,
insert NULL
                        n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                        "session_id_one_previous = session_id_one, session_id_one = NULL
"
                                        "WHERE tunnel_side_id = %s AND tunnel_side_server_id = %s",
                                        row[4], row[5]);
                }
                free(inode);

                //flockfile(stdout);
                //fprintf(stdout, "%s\n", sql_query);
                //funlockfile(stdout);

                if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
                        flockfile(stderr);
                        fprintf(stderr, "%s: Failed executing query to update tunnel_side: Error: %s\n",
timestamp, mysql_error(mysql_connection));
                        funlockfile(stderr);
                        free(sql_query);
                } else {
                        free(sql_query);
                }

                 } else if(strcmp(row[15], "ips_emulated")==0) {
                        assert(start_oid = malloc(200));

                //walk the phase 1 oid
                snprintf(start_oid, 200, "hrSWRunEntry.hrSWRunIndex");
                ips_snmpwalk(device, row[15], start_oid, "25977", &index_queue, 1);
                free(start_oid);

                //process the queue in order to get the phase 1 id
                int count = 0;
                while(index_queue.head != NULL) {
                                if(inode > 0)
                                free(inode);
                        inode = (index_node *) queue_get(&index_queue);
                        ++count;
                }

                //update the phase 1 id in db...
                assert(sql_query = malloc(800));
                if(count > 0) {
                        n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                        "session_id_one_previous = session_id_one, session_id_one = %d "
                                        "WHERE tunnel_side_id = %s AND tunnel_side_server_id = %s",
                                                inode->value, row[4], row[5]);
```

```
                                free(inode);
                        } else {
                                //there was nothing in the queue, so we know we didn't find a matching session id,
insert NULL
                                n=snprintf(sql_query, 800, "UPDATE tunnel_side SET "
                                        "session_id_one_previous = session_id_one, session_id_one = NULL
"
                                        "WHERE tunnel_side_id = %s AND tunnel_side_server_id = %s",
                                        row[4], row[5]);
                        }

                        //flockfile(stdout);
                        //fprintf(stdout, "%s\n", sql_query);
                        //funlockfile(stdout);

                        if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
                                flockfile(stderr);
                                fprintf(stderr, "%s: Failed executing query to update tunnel_side: Error: %s\n",
timestamp, mysql_error(mysql_connection));
                                funlockfile(stderr);
                                free(sql_query);
                        } else {
                                free(sql_query);
                        }

                        assert(start_oid = malloc(200));
                        //walk the phase 2 oid
                        snprintf(start_oid, 200, "hrSWRunEntry.hrSWRunName");
                                ips_snmpwalk(device, row[15], start_oid, "\"apache\"", &index_queue, 0);
                                free(start_oid);

                        //find all cisco phase 2 monitors related to this tunnel side so they can be updated
                        assert(sql_query = malloc(1000));
                        n = snprintf(sql_query, 1000, "SELECT monitor.monitor_id, monitor.monitor_server_id "
                                                "FROM monitor_tunnel, monitor, metric, metric_snmp "
                                        "WHERE monitor_tunnel.tunnel_side_id=%s AND
monitor_tunnel.tunnel_side_server_id=%s "
                                        "AND monitor.monitor_id=monitor_tunnel.monitor_id "
                                        "AND monitor.monitor_server_id=monitor_tunnel.monitor_server_id "
                                        "AND metric.metric_id=monitor.metric_id "
                                        "AND metric.metric_server_id=monitor.metric_server_id "
                                        "AND metric.suite='snmp' "
                                        "AND metric_snmp.metric_id=metric.metric_id "
                                        "AND metric_snmp.metric_server_id=metric.metric_server_id "
                                        "AND metric_snmp.phase = 'two' ",
                                                row[4], row[5]);

                                if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
                                flockfile(stderr);
                                fprintf(stderr, "%s: Failed executing query for phase 2 monitors: Error: %s\n",
timestamp, mysql_error(mysql_connection));
                                funlockfile(stderr);
                                free(sql_query);
                        } else {
                                free(sql_query);
                        }
```

```
MYSQL_RES *phase_two_result;
MYSQL_ROW monitor_row;
phase_two_result = mysql_store_result(mysql_connection);

    while(monitor_row=mysql_fetch_row(phase_two_result)) {
char *sql_query_delete;
int len;
assert(sql_query_delete = malloc(8000));
//additional where clauses will be appened as we go to avoid deleting the rows that
are still in use...
len = snprintf(sql_query_delete, 8000, "DELETE FROM
monitor_tunnel_cisco_phase_2 WHERE monitor_id=%s AND monitor_server_id=%s", monitor_row[0],
monitor_row[1]);

        inode = (index_node *) index_queue.head;
while(inode != NULL) {
            //check if there is already a row for this session id value
            assert(sql_query = malloc(1000));
    n=snprintf(sql_query, 1000, "SELECT COUNT(*) AS count FROM
monitor_tunnel_cisco_phase_2 "
            "WHERE monitor_id=%s AND monitor_server_id=%s "
                "AND phase_2_id=%d ", monitor_row[0],
monitor_row[1], inode->value);

            MYSQL_RES *count_result;
            MYSQL_ROW count_row;
            int row_count;

    if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
        flockfile(stderr);
                fprintf(stderr, "%s: Failed executing query for phase 2
row count: Error: %s\n", timestamp, mysql_error(mysql_connection));
        funlockfile(stderr);
                free(sql_query);
                row_count = 0;
    } else {
        free(sql_query);
                count_result = mysql_store_result(mysql_connection);
            count_row = mysql_fetch_row(count_result);
        row_count = atoi(count_row[0]);
            mysql_free_result(count_result);
                }
    if(row_count < 1) {
                //there is no row for this session id, create one
                assert(sql_query=malloc(800));
        n=snprintf(sql_query, 800, "INSERT INTO monitor_tunnel_cisco_phase_2 "
                    "(monitor_id, monitor_server_id, phase_2_id, counter,
timestamp) "
                "VALUES (%s, %s, %d, 0, 0)",
                    monitor_row[0], monitor_row[1],
inode->value);
        if(mysql_real_query(mysql_connection, sql_query, n)!=0) {
            flockfile(stderr);
                fprintf(stderr, "%s: Failed executing query to
insert into monitor_tunnel_cisco_phase_2: Error: %s\n", timestamp, mysql_error(mysql_connection));
            funlockfile(stderr);
```

```
                                        free(sql_query);
                                } else {
                                        free(sql_query);
                                }
                                                }
                                                //update the delete query to not delete this row (since it still is in
use)
                                char *temp_string;
                                temp_string = strdup(sql_query_delete);
                                len = snprintf(sql_query_delete, 8000, "%s AND NOT(phase_2_id=%d) ",
temp_string, inode->value);
                                free(temp_string);
                                inode = (index_node *)inode->next;
                                        }
                        //execute the delete for all rows for this monitor that were not still in use
                        if(mysql_real_query(mysql_connection, sql_query_delete, len)!=0) {
                                flockfile(stderr);
                                                fprintf(stderr, "%s: Failed executing query to delete old phase 2
rows: Error: %s\n", timestamp, mysql_error(mysql_connection));
                                funlockfile(stderr);
                                free(sql_query_delete);
                        } else {
                                free(sql_query_delete);
                        }
                             }
                             //cleanup the queue
                             while(index_queue.head != NULL) {
                                        inode = (index_node *) queue_get(&index_queue);
                                        free(inode);
                             }
                mysql_free_result(phase_two_result);


                 }
                 //...
        }
        mysql_free_result(result);

        return(0);
}
```